

Auto-Encoding Variational Bayes

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

Reviewer: 이진모

cf) 본 논문은 VAE의 구조적인 특징보다, Variational Bayes 방법으로 확률분포를 근사추론하는 수식적인 방법을 주로 다룬다. 따라서 본 문서에서는 위 논문과 함께 VAE를 이해하는 데 필요한 다양한 개념을 종합하여 설명하고자 시도한다.

1. Backgrounds

Variational Auto Encoding(이하 VAE)를 이해하기 앞서 먼저 공부해야 할 개념이 있으니, 바로 Representation Learning과 Generative modeling에 대한 개념이다.

1.1. Representation Learning

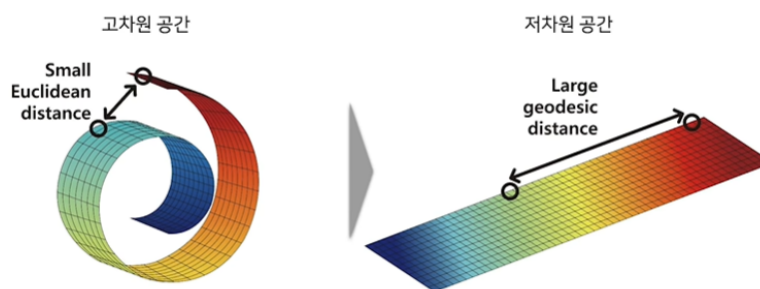
Representation Learning(이하 표현 학습)에서 이야기하는 인사이트는 다음의 두 가지이다.

1. 우리에게 주어지는 데이터는 그 표현 방식에 따라 다양한 차원으로 나타낼 수 있다.
2. 실제로 데이터가 관측되고 생성되는 데 필요한 부분은 전체 차원 중 매우 작은 subset일 뿐이다.

설명만 들어서는 쉽게 이해되지 않는다. 예시를 통해 알아보자.

구불구불한 산길을 지나가는 자동차를 상상해보자. 인공위성은 위도와 경도, 고도를 가지고 자동차의 위치를 3차원 벡터로 나타낼 수 있다. 그러나 그 자동차 안에서 우리의 네비게이션은 기준점으로부터의 거리, 즉 1차원 벡터를 통해 자동차의 위치를 나타낸다. 따라서 동일한 데이터가 다양한 차원에서 표현될 수 있다.

또한 실세계의 데이터는 매우 고차원의 공간에서 정의되는데, 그 전체 공간 중 데이터가 존재하는 공간은 매우 작은 일부분일 뿐이다. 이러한 현상을 잘 설명하는 가설이 **Manifold 가설**이다. Manifold 가설은 우리의 관측 데이터가 존재하는 고차원의 공간 속에, 데이터가 아주 잘 밀집되어 있는 저차원인 비선형 공간이 존재한다고 설명한다.



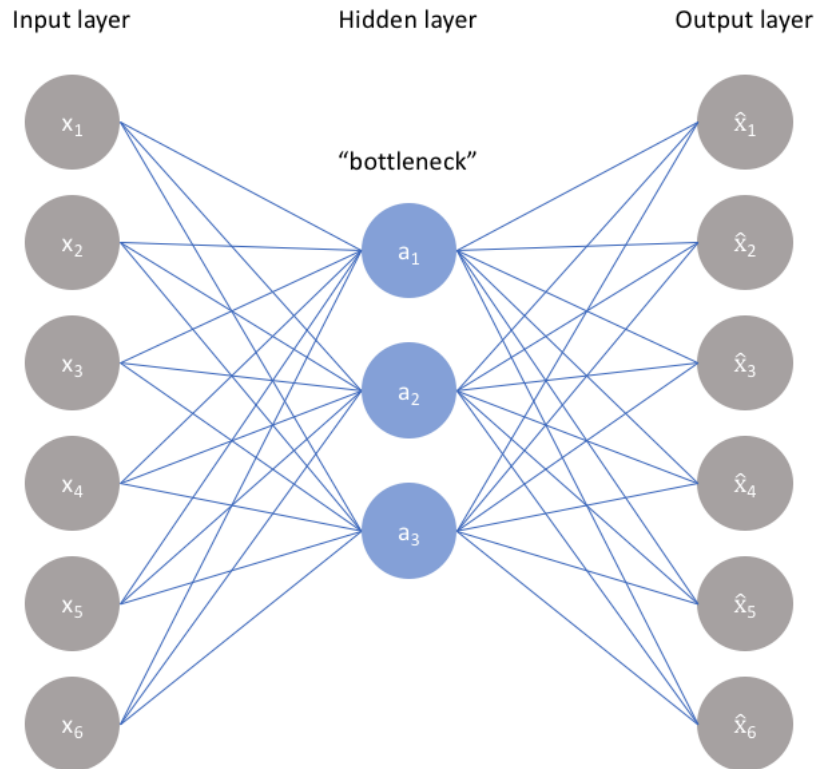
고차원의 공간에서 정의되는 좌측과 같은 스위스 롤 형태의 데이터는 사실 그 고차원 공간 중 매우 일부분에만 존재하며, 이를 저차원의 공간 상에서 다른 형태로 표현할 수 있다는 것이 매니폴드 가설의 핵심이다. 이 때 저차원의 공간에서 비선형의 형태로 데이터가 새롭게 표현되는 부분공간을 매니폴드라고 한다. 이러한 매니폴드는 우리가 실제로 관측하는 데이터로는 표현할 수 없고, 다만 관측되는 변수들의 복잡한 비선형 변환에 의해서 정의되는 변수들로 구성되는데, 이러한 변수를 '잠재 변수(Latent variable)'라고 한다.

우리는 사실 지도학습을 수행하며 이러한 매니폴드 학습을 자연스럽게 사용하게 된다. 예를 들어서 인공신경망의 구조를 생각해보자. 인공신경망에서는 한 층의 출력값이 다음 층의 입력이 되는데, 이렇게 층을 지날 때마다 비선형 변환이 수행된다. 이를 다시 표현하면 입력벡터의 실제 공간을 해당 층의 출력노드 개수만큼의 차원을 갖는 새로운 공간으로 변환하는 거라 볼 수 있다. 따라서 인공신경망의 훈련이란, **가장 마지막 층의 출력 공간이 관측 데이터의 분포와 특성을 가장 잘 나타내는 공간상에서 표현되도록 데이터에 내재된 매니폴드를 학습하는 훈련인 셈이다.** 이러한 학습을 표현학습이라 한다.

1.2. Auto-Encoder

오토 인코더는 표현학습이 가능하도록 디자인한 NN 모듈이다. 오토 인코더의 목적은 입력된 데이터의 압축된 표현(compressed knowledge representation)을 통해 입력 데이터가 재표현된 출력을 만들어내는 것이고, 이 목적을 달성하기 위해 layer 사이에 bottleneck layer(이하 병목층)를 두는 신경망 구조를 갖는다.

만약 입력된 데이터 간의 아무런 의존성이 없다면 이러한 압축된 표현을 만들어내는 것은 힘들겠지만 어떠한 형태라도 데이터 상의 구조적인 특징이 있다면 병목층에서 이러한 구조를 포착해 낼 수 있다.



Source: <https://www.jeremyjordan.me/autoencoders/>

위 그림과 같은 신경망 구조에 라벨이 없는 데이터 \mathbf{x} 를 넣고, 그 입력의 새로운 표현(reconstruction이라고 원문은 표현)인 $\hat{\mathbf{x}}$ 를 출력으로 하는 지도학습을 수행하는 모듈이 바로 오토 인코더인 셈이다. 이 때 학습은 재구성 에러인 $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ 을 최소화하는 방향으로 이루어진다.

그런데 왜 이런 의미없는 일을 할까? 입력과 출력 사이의 재구성 에러를 최소화하는 방향으로 학습하기 위해 굳이 병목층까지 들어간 복잡한 모델을 만들 필요가 있을까? 이를 다시 생각해 보면, 오토 인코더의 목적은 입력과 가장 닮은 출력을 만들어내는 것뿐만이 아니라는 뜻이다. 관련 문서를 읽어보며 드는 생각은, 오토 인코더의 목적은 latent space를 찾아내는 것이다. 조금 더 정확히 표현하면 original input을 가장 잘 담고 있는 latent space를 찾았다가, 그것을 다시 original domain만큼 펼쳤을 때 데이터 손실이 가장 적은 모델을 만드는 것이다.

따라서 오토 인코더의 병목층은 매우 중요하다. 데이터는 병목층을 지나며 다음과 같은 목표를 추구해야만 한다.

1. 정확한 재구성을 가능하게 하도록 입력 값을 충실히 학습할 것
2. 재구성이 단순한 입력의 암기가 되지 않도록 적당히 다를 것

따라서 오토 인코더의 진짜 손실함수는 재구성 에러에 규제 term이 더해진 형태이다.

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) + \text{regularizer}$$

이러한 오토 인코더의 활용이나 존재 이유에 대해서는 조금 더 공부한 후 추가할 예정,,,

1.3. Generative Modeling

우리에게 익숙한 모델링의 구조, 즉 데이터 x 가 주어지고 타겟 y 를 위한 분류나 회귀를 수행하는 모델은 Discriminative model, 분별모델이다. 그러나 Generative Model(이하 생성모델)은 데이터 x 가 주어졌을 때 그것의 분포 $P(x)$ 를 추정하는 것을 목표로 한다. 정확히 표현하면 그러한 분포를 파라미터로 나타내는 $P(x; \theta)$ 를 학습하는 것이 생성모델의 목적이다.

이러한 생성모델은 데이터의 밀도함수를 미리 정의하고 추정하는 방식과 정의 없이 추정하는 방식으로 나뉜다. 전자를 explicit, 후자를 implicit 이라고 표현하는데, explicit한 방법 중에서도 tractable한 밀도함수를 가정하여 직접 추정하는 방식과 intractable하여 근사할 수 밖에 없는 밀도함수를 추정하는 방식으로 나뉜다. VAE는 Variational Inference(이하 변분추론)를 이용해 데이터의 밀도함수를 근사 추정하는 생성모델을 만든다. 그런데 그 과정을 인코더와 디코더로 나누어 수행하다보니 형태가 AE와 동일해져, VAE라는 이름이 붙은 것이다.

그렇다면 결국에는 $P_{data}(x; \theta)$ 와 가장 근접한 $P_{model}(x; \theta)$ 를 모델링해야 하는데, 이것을 직접 최적화할 수가 없다. 그 이유는 대부분의 P_{model} 은 intractable한 형태이기 때문이다. 따라서 우리는 직접 P_{model} 을 모델링하는 대신, x 에 많은 영향을 미치는 잠재변수 z 를 가정하고 $z \rightarrow x$ 의 관계, 즉 $P(x|z)$ 를 모델링하기로 한다. 이러한 모델링이 Latent variable Modeling이고, VAE는 위 방법을 따른다.

본 문서에서 다루는 논문에서는 이러한 잠재변수가 continuous하고 intractable한 사후분포를 갖고 있을 때 variational lower bound(이하 설명 예정)를 reparameterization 하여 lower bound의 미분가능한 unbiased estimator를 효율적으로 찾는 방법을 제안한다. 이 방법을 Stochastic Gradient Variational Bayes(SGVB)라고 한다.

2. Method

결국 우리의 목표는 $P(\mathbf{x}|\mathbf{z})$ 의 모델링이다. 그러나 z 는 잠재변수이기 때문에 이 정보를 가지고 직접 모델링이 불가능하다. 따라서 첫 번째 트릭이 등장한다. \mathbf{z} 에도 분포가 있다고 가정하고, 그 분포를 추정하여 $P(\mathbf{z})$ 로부터 \mathbf{z} 를 샘플링한 뒤 그것을 이용하여 $P(\mathbf{x}|\mathbf{z})$ 를 학습시키는 트릭이다.

그러나 \mathbf{z} 의 분포를 추론하는 것 역시 쉽지 않다. 따라서 논문 저자는 데이터 \mathbf{x} 가 주어졌을 때 사후확률 $P(\mathbf{z}|\mathbf{x})$ 가 있다고 가정하고, 이로부터 \mathbf{z} 의 사후표본을 샘플링 해 $P(\mathbf{x}|\mathbf{z})$ 에 전달하는 두 번째 트릭을 사용한다. 따라서 VAE의 학습과정을 간단하게 요약하면 다음과 같다.

1. \mathbf{x} 를 입력으로 받아 $P(\mathbf{z}|\mathbf{x})$ 를 추론한다. $P(\mathbf{z}|\mathbf{x})$ 는 인코더가 되어 $\mathbf{x} \rightarrow \mathbf{z}$ 의 매핑을 담당한다.
2. 이로부터 \mathbf{z} 를 샘플링한 후 $P(\mathbf{x}|\mathbf{z})$ 에 전달해 학습시킨다. $P(\mathbf{x}|\mathbf{z})$ 는 디코더가 되어 $\mathbf{z} \rightarrow \mathbf{x}$ 의 매핑을 담당한다.

그러나 아직도 해결되지 않은 문제가 있다.

- Intractability: 사후분포인 $P_\theta(\mathbf{z}|\mathbf{x}) = P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})/P_\theta(\mathbf{x})$ 가 intractable하기 때문에 marginal likelihood인 $P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{z})P_\theta(\mathbf{x}|\mathbf{z})d\mathbf{z}$ 역시 intractable하여 EM 알고리즘을 통한 lower bound 직접 추론이 불가능한 상황.
- Large Dataset: 데이터의 크기가 너무 커 미니배치를 통한 파라미터 업데이트의 비용이 지나치게 큰 데이터셋

따라서 이 섹션에서는 SGVB를 이용하여 $P_\theta(\mathbf{x})$ 의 lower bound를 도출하고 그것을 최대화하는 과정을 보여준다. 제한하는 가정은 i.i.d 데이터셋에 대해 각 데이터포인트마다 잠재 변수가 존재하고, global parameter에 대해서는 ML 추론을, 잠재 변수에는 변분추론을 행한다는 것뿐이다.

그런데 $P_\theta(\mathbf{x})$ 를 구하는 데 필요한 사후분포 $P_\theta(\mathbf{z}|\mathbf{x})$ 역시 intractable하기 때문에, 이를 잘 근사하는 사후분포인 $q_\phi(\mathbf{z}|\mathbf{x})$ 를 이용하는 변분추론을 수행한다. $q_\phi(\mathbf{z}|\mathbf{x})$ 를 이용해 $P_\theta(\mathbf{x})$ 의 lower bound를 구하는 과정은 아래와 같다. 모든 계산은 분포에 로그를 취해 진행한다.

$$\begin{aligned} \log p_\theta(\mathbf{x}^{(i)}) &= E_z \left[\log \frac{p_\theta(\mathbf{x}^{(i)}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x}^{(i)})} \right] = E_z \left[\log \frac{p_\theta(\mathbf{x}^{(i)}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x}^{(i)})} \frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} \right] \\ &= E_z \left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) \right] - E_z \left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_\theta(\mathbf{z})} \right] + E_z \left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_\theta(\mathbf{z}|\mathbf{x}^{(i)})} \right] \\ &= E_z \left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) \right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) \end{aligned}$$

첫 번째 등호는 $p_\theta(\mathbf{x}^{(i)})$ 이 z 와 독립이라는 가정 하에 성립한다. 이렇게 decompose된 수식을 하나씩 뜯어보자.

$$E_z \left[\log p_\theta(x^{(i)}|z) \right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))$$

첫 번째 항은 z 를 샘플링하는 함수의 로그 가능도를 의미한다. 즉, 샘플링된 z 로부터 x 가 얼마나 잘 복원되었는가를 의미한다. 두 번째 항은 prior probability $p_\theta(z)$ 와 $q_\phi(z|x^{(i)})$ 의 차이를 의미한다. 이 차이는 당연히 작을수록 좋다. 마지막 항은 intractable한 p_θ 와 이를 근사하기 위해 가정한 q_ϕ 의 차이를 의미한다. 그러나 이 차이는 직접 구하는 것이 불가능하다. 따라서 마지막 항은 두 채로 앞의 두 항을 최대화 함으로써 $\log p_\theta(\mathbf{x}^{(i)})$ 를 최대한 근사한다. 즉, 첫 번째와 두 번째 항이 ELBO가 된다.

위 식의 가장 아랫줄 수식의 첫 번째 두 번째 항을 $\mathcal{L}(x^{(i)}, \theta, \phi)$ 로 바꿔서 식을 전개하면 아래와 같다.

$$\log p_\theta(x^{(i)}) = \mathcal{L}(x^{(i)}, \theta, \phi) + D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))$$

그런데 K-L Divergence의 값은 항상 양수이므로 다음의 부등식이 성립한다.

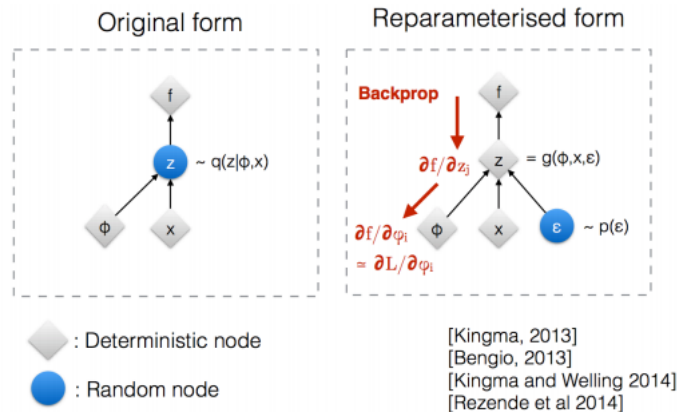
$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

우리는 이를 최대화 함으로써 tractable한 \mathcal{L} 로 intractable한 사후분포를 추정할 수 있게 되었다. 그런데 문제가 하나 있다.

$$E_z \left[\log p_\theta(x^{(i)}|z) \right]$$

최적화 해야 하는 \mathcal{L} 의 항 중 위와 같은 항이 있는데, 이를 보면 z 를 샘플링 해서 계산을 해야 한다. 이 때 z 는 $q_\phi(z|x^{(i)})$ 를 따른다(고 가정했다). 그리고 이 과정을 NN 모델로 풀어내고자 하는 것이 VAE이다. 그런데 어떠한 분포로부터 샘플링하는 과정은 미분으로 정의되는 과정이 아니다. 즉 gradient를 구할 수가 없다. 즉 backpropagation이 불가능하다. NN 모델에서 gradient descent로 문제를 해결한다는 말은 즉 그 모델이 어떤 파라미터들에 대해 미분이 가능해야 하고, 그 말은 다시 모델이 deterministic 하다는 뜻인데, 샘플링이라는 과정은 애초에 stochasticity가 포함된 개념이기 때문에 이 과정이 불가능한 것이다. 따라서 이를 해결하기 위해 논문에서는 reparameterization trick을 소개하며, 이것이 SGVB의 개념이다.

3. Reparameterization trick



정규분포 $\mathcal{N}(\mu, \sigma^2)$ 으로부터 추출된 표본 z 는 수학적으로 $\mu + \sigma * \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$ 와 동일하다. 이처럼 (auxiliary) noise variable을 추가해 분포를 다르게 표현해줌으로써 우리는 원하는 파라미터에 대한 그라디언트를 계산할 수 있다. 정규분포 $\mathcal{N}(\mu, \sigma^2)$ 으로부터 추출된 표본 z 에 대해서 $\frac{\partial z}{\partial \mu}$, $\frac{\partial z}{\partial \sigma}$ 는 계산할 수 없지만, $\mu + \sigma * \epsilon$ 에서 해당 계산을 할 수 있는 것처럼 말이다. 따라서 논문에서는 다음과 같은 reparameterization을 제안한다.

$$z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)}), \quad \text{where } \epsilon^{(i,l)} \sim p(\epsilon)$$

위의 새로운 표현을 사용해 \mathcal{L} 의 두 항 중 $E_z [\log p_\theta(x^{(i)}|z)]$ 를 다시 표현하여 전체 \mathcal{L} 을 미분 가능하게 만들고, 이로부터 그 값을 최적화하여 ELBO를 최대화하면 $\log p_\theta(\mathbf{x}^{(i)})$ 가 근사되고, 이로부터 $P(\mathbf{z}|\mathbf{x})$ 가 계산된다. 이것이 SGVB 최적화이다.

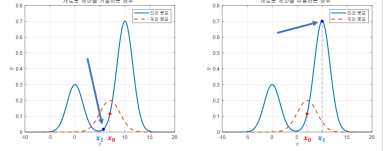
임의의 함수 $f(z)$ 의 $q_\phi(z|x)$ 에 대한 MCMC estimate은 다음과 같다고 한다.

▼ MCMC estimate

Markov Chain Monte Carlo

Interactive MCMC JS Applet by Chi-Feng. 소스코드 이 포스팅에 대해 잘 이해하기 위해선 다음의 내용에 대해 알고 오시는 것이 좋습니다. 위키피디아에 따르면 마르코프 연쇄 몬테카를로 방법(Markov Chain Monte Carlo, MCMC)은 "마르코프 연쇄의 구성에 기반한 확률 분포로부터 원하는 분포의 정적 분포를 갖는 표본을 추출하는

<https://angeloyeo.github.io/2020/09/17/MCMC.html>



$$\mathbb{E}_{q_\phi(z|x^{(i)})} [f(z)] = \mathbb{E}_{p(\epsilon)} [f(g_\phi(\epsilon, x^{(i)}))] = \frac{1}{L} \sum_{l=1}^L f(g_\phi(\epsilon^{(l)}, x^{(i)})),$$

where $\epsilon^{(l)} \sim p(\epsilon)$

따라서 전체 \mathcal{L} 의 reparameterized term은 다음과 같다.

$$\tilde{\mathcal{L}}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + \frac{1}{L} \sum_{l=1}^L (\log p_\theta(x^{(i)}|z^{(i,l)})),$$

where $z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$

그런데, z 에 대한 term만 reparameterize 하면 된다는 것은, K-L Divergence term은 deterministic 하다는 말인가? 그렇다. 이에 대한 증명이 Appendix B. 에 의해 논문에 제시되어 있다.

▼ Appendix B

Suppose,
 $p_\theta(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$ & $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ is gaussian.

Then,

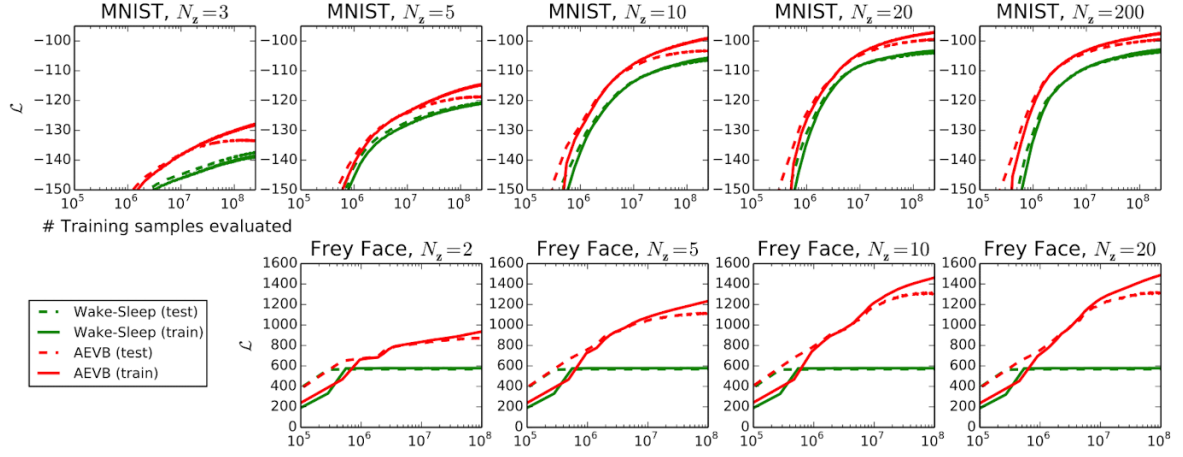
$$\begin{aligned} \int q_\phi(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) d\mathbf{z} = -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 + \sigma_j^2) \\ \int q_\phi(\mathbf{z}) \log q_\phi(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) d\mathbf{z} = -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (1 + \log \sigma_j^2) \\ \therefore -D_{KL}((q_\phi(\mathbf{z})||p_\theta(\mathbf{z}))) \\ &= \int q_\theta(\mathbf{z}) (\log p_\theta(\mathbf{z}) - \log q_\theta(\mathbf{z})) d\mathbf{z} = \frac{1}{2} \sum_{j=1}^J (1 + \log((\sigma_j^2)) - (\mu_j)^2 - (\sigma_j)^2) \end{aligned}$$

이에 더해, 논문에서는 $q_\phi(\mathbf{z}|\mathbf{x})$ 의 형태에 따라 적절한 미분가능한 transformation $g_\phi(\epsilon, \mathbf{x})$ 와 $\epsilon \sim p(\epsilon)$ 을 찾는 방법을 제시하고 있다.

1. Tractable ICDF: 지수분포, 코시분포, 로지스틱분포, 레일리분포 등 ICDF가 tractable한 분포로 q_ϕ 를 상정한 경우 그 ICDF를 g_ϕ 로 두면 되고, 이 때 noise variable ϵ 은 $\mathcal{U}(0, \mathbf{I})$ 를 따르도록 설정한다.
2. Location-scale family: 정규분포, 로지스틱분포, 삼각분포 등 location-family인 분포로 q_ϕ 를 상정했다면, ϵ 은 location-scale family의 분포 중 location = 0, scale = 1인 표준 분포를 따르게 하면 되고, 이 때 g_ϕ 는 $location + scale \cdot \epsilon$ 으로 한다.

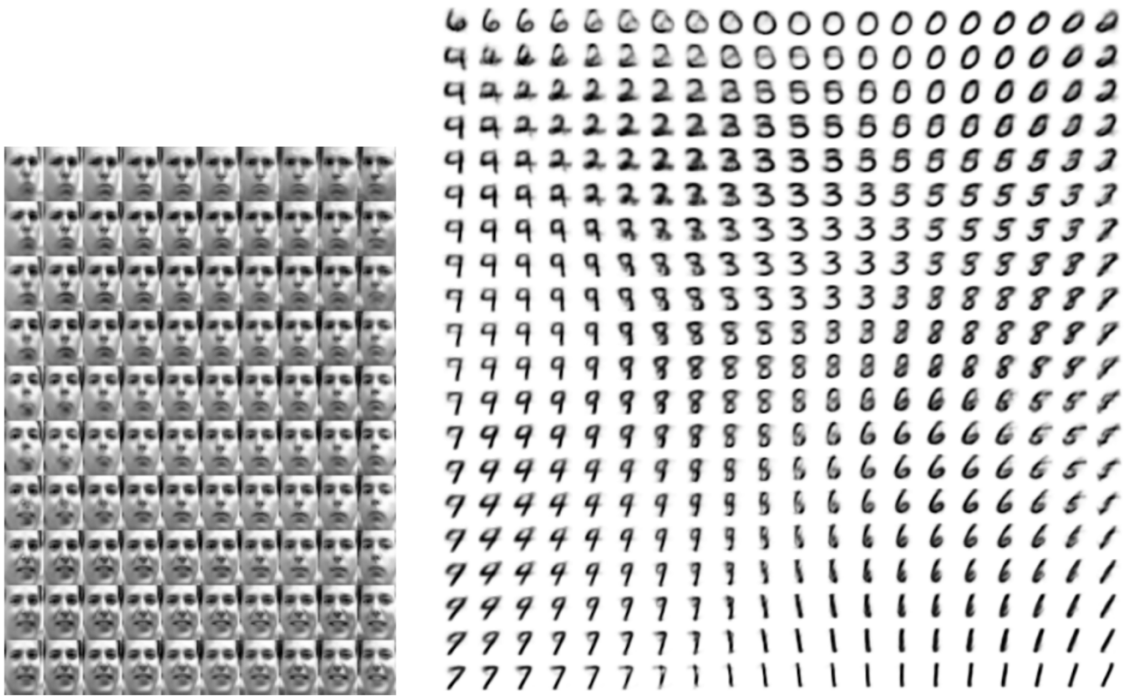
4. Experiments

논문에서는 MNIST 데이터와 Frey Face 데이터를 가지고 당시의 SOTA인 'wake-sleep algorithm'과의 variational lower bound에 대한 비교가 이루어졌다. 얼마나 빨리 ELBO가 수렴하는지, 그리고 얼마나 최대화되는지에 대한 비교(인 것 같다.)가 진행되었다.



N_n 은 latent vector의 차원 수, x축은 사용된 데이터 instance의 갯수, y축은 lower bound \mathcal{L} 이다. 모든 차원의 잠재변수벡터에 대하여 SOTA보다 SGVB를 활용한 최적화가 더 빠르게 수렴하고 또 더 근사하게 수렴하는 사실을 발견할 수 있다. 그말은 즉 z 로부터 x 를 더욱 잘 추출할 수 있다는 의미가 된다.

만약 잠재변수의 차원을 2차원으로 둔다면 2차원 평면상에 z 로 이루어진 manifold를 시각화해볼 수 있다.



(a) Learned Frey Face manifold

(b) Learned MNIST manifold

그림에서 볼 수 있듯이 latent variable로 만들어진 manifold가 실제로 유사한 특징들에 대한 정보를 담고 있는 것을 확인할 수 있다. (매우 놀랍다)

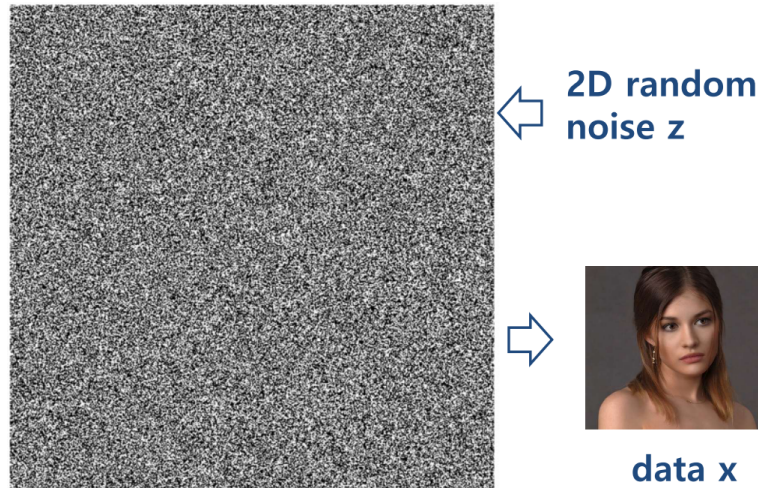
5. Appendix

공부를 하면서 VAE가 그래서 정확히 무엇을 하는 모델인지에 대한 직관적인 이해를 하기가 어려웠다. 그리고 위에 정리된 문서를 읽어도 어렵듯한 개념만 이해될뿐 선명하게 VAE의 개념이 와닿지 못할 것 같다. 따라서 부록을 통해 VAE의 직관적인 이해를 도우려 한다. 먼저

다시 AE로 돌아가보자.

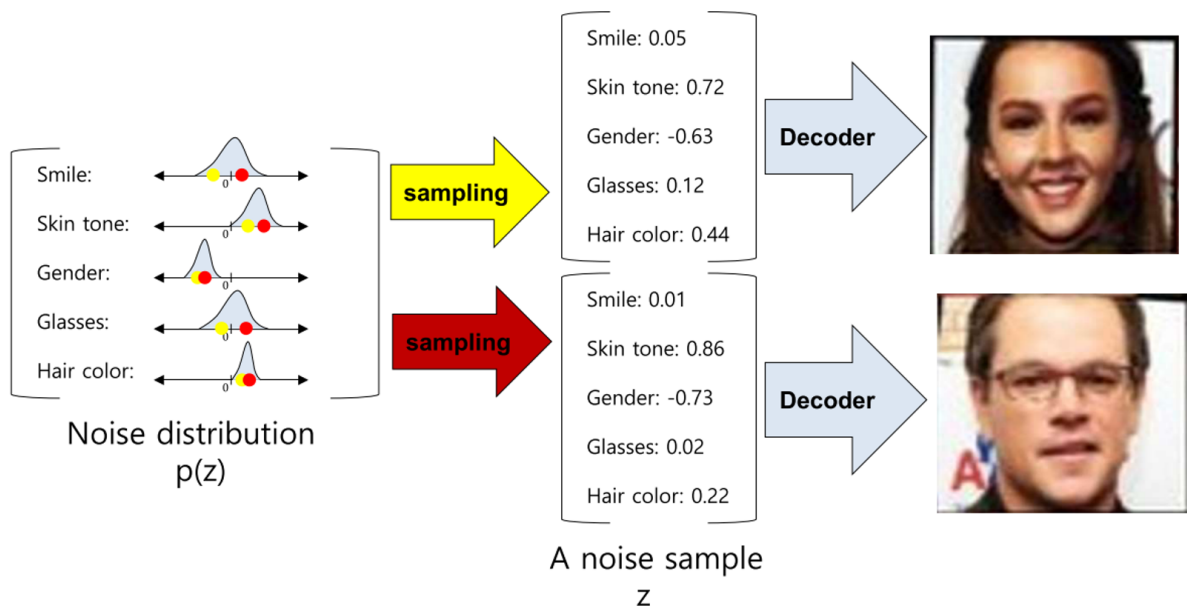
AE는 인코더에서 원본 x 로부터 저차원 매니폴드 상의 z 를 구하고, 디코더가 이를 받아 다시 원본 x 와 비슷한 \hat{x} 를 만든다. 결국 AE의 목적은 x 와 \hat{x} 가 최대한 비슷하게 하는 것이며, 인코더는 이를 위한 x 의 압축, 디코더는 압축된 $x(=z)$ 의 복원을 맡는다.

VAE는 AE와 구조적으로 비슷하지만 그 목적이 다르다. VAE는 랜덤한 노이즈로부터 원하는 이미지를 얻을 수 있는지에 대한 의문에서 시작되었다고 한다.



그러나 상식적으로 이러한 랜덤 노이즈로부터 원하는 이미지를 추출하는 것은 매우 어렵다. 그런데 만약 랜덤한 노이즈가 아니라 어떠한 분포를 따르는 노이즈라면 어떨까? 만약에 $P_{data}(x)$ 를 따르는 데이터셋이 있어서 이러한 데이터셋의 original 분포를 잘 나타내는 P_{model} 을 만들고, 이로부터 랜덤값을 생성한다면, 주어진 데이터셋과 유사한 값을 뽑아낼 수 있을 것이다.

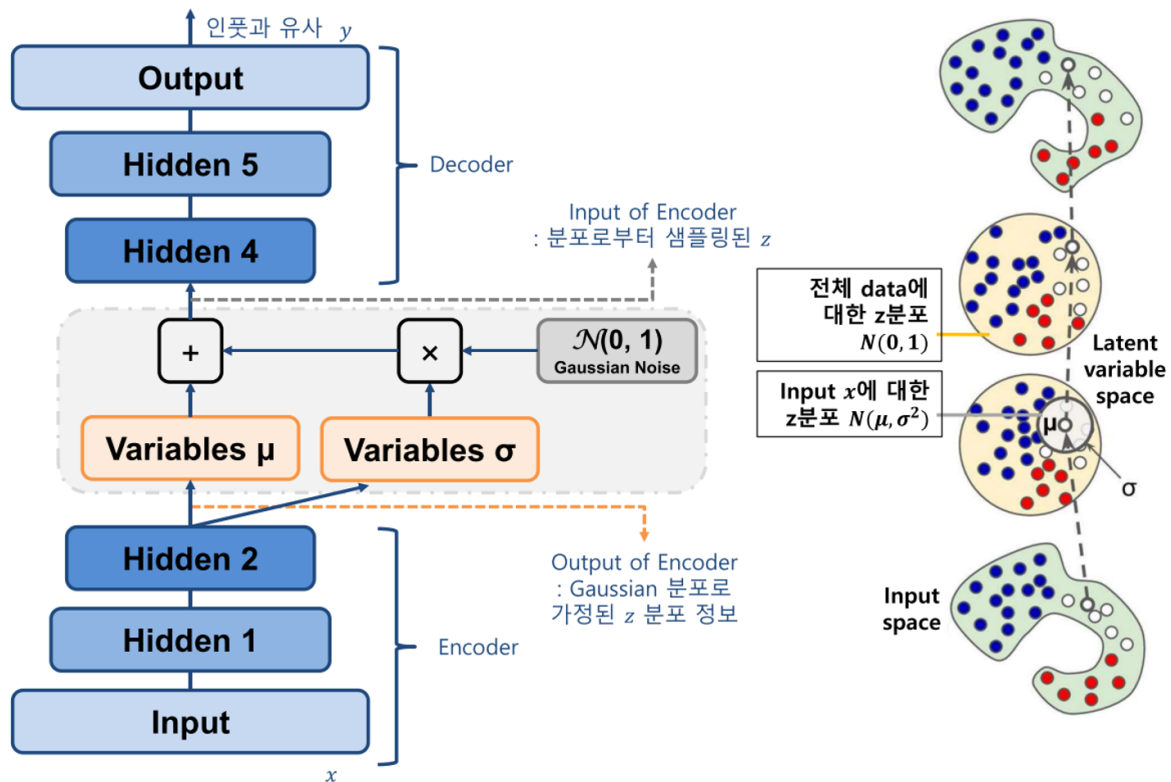
그렇기 때문에 AE의 목적이 병목증을 통한 차원의 변화를 통해 잠재 변수 z 를 찾아내는 것이었다면, VAE의 목적은 원본 데이터와 유사한 데이터 z 를 생성하기 위해 그것의 분포 자체를 학습하는 것이다. 그리고 이 분포로부터 샘플링 된 z 를 디코더의 입력으로 받게 된다.



VAE가 데이터를 입력으로 받으면 인코더로부터 $P(z)$ 가 찾아지고, 이로부터 샘플링 된 z 는 더 이상 랜덤하지 않으며 그 분포의 특징을 잘 갖는 출력으로 나타날 것이다. 위 그림이 이 과정을 잘 설명한다(실제로 잠재변수가 예시처럼 explicit하게 찾아지는 것은 아니다). 이 때 $P(z)$ 를 직접 찾는 것이 어려우므로 $P(z|x)$ 를 찾기로 하고, 이 과정에서 $P_\theta(z|x)$ 역시 intractable하므로 이것과 근사한 $q_\phi(z|x)$ 를 사용

해 결국 우리가 찾고싶은 분포인 $P(x)$ 의 ELBO를 찾아 최대화 하는 변분추론의 개념이 적용된다. 이제 조금 위에서 설명한 개념의 조각이 맞춰지는 듯 하다.

그런데 VAE역시 딥러닝 모듈이기 때문에 역전파 학습이 가능해야 하고, 역전파 학습은 알다시피 그라디언트를 사용해 이루어진다. 그런데 VAE의 학습과정에서 추론한 $P(z)$ 로부터 z 를 샘플링하는 과정, 즉 deterministic하지 못한 과정이 필연적으로 포함되기 때문에 이 과정에서 그라디언트 계산이 불가능해진다. 따라서 이를 해결하기 위해 reparameterization trick을 사용해 z 를 샘플링 하는 과정역시 미분이 가능하도록 조치를 취한 것이다.



위 그림에서 회색 박스에 표현된 부분이 Reparameterization part이다. 예시에서는 $\mathcal{N}(0, 1)$ 을 따르는 noise를 만들었지만 이 과정은 위에서 제시한 2가지 방법에 따라 다양한 Variation이 있을 수 있다.

6. Code Realization

MNIST 데이터를 활용해 Pytorch framework로 2차원 latent space를 만들고 다시 데이터를 generate하는 모듈을 코드로 구현해보았다. 코딩링크는 하단에 남기도록 하겠다. 주석을 잘 읽어보면 논문 내용과 모듈 구조가 쉽게 연결될 것이다.

```
class VAE(nn.Module):
    def __init__(self, image_size, hidden_size_1, hidden_size_2, latent_size):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(image_size, hidden_size_1)
        self.fc2 = nn.Linear(hidden_size_1, hidden_size_2)
        self.fc31 = nn.Linear(hidden_size_2, latent_size)
        self.fc32 = nn.Linear(hidden_size_2, latent_size)
        # fc31/fc32는 hidden_size2로부터 latent size만큼의 latent vector를 위한 mu와 sigma를 뱉어낸다.

        self.fc4 = nn.Linear(latent_size, hidden_size_2)
        self.fc5 = nn.Linear(hidden_size_2, hidden_size_1)
        self.fc6 = nn.Linear(hidden_size_1, image_size)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        h2 = F.relu(self.fc2(h1))
        return self.fc31(h2), self.fc32(h2)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + std * eps # Reparameterize 함수를 선언함으로써 z를 reparameterize 한다.
```



```

def decode(self, z):
    h3 = F.relu(self.fc4(z))
    h4 = F.relu(self.fc5(h3))
    return torch.sigmoid(self.fc6(h4)) # 이미지와 동일한 차원을 가지면서 시그모이드를 통과해 (0,1)의 값을 갖는다.

def forward(self, x):
    mu, logvar = self.encode(x.view(-1, 784)) #sigma를 logvar로 받는 이유는 표준편차가 음수가 되지 않기 위해 log(sigma^2)꼴로 생각하기 위함이다.
    z = self.reparameterize(mu, logvar)
    return self.decode(z), mu, logvar

VAE_model = VAE(28*28, 512, 256, 2).to(DEVICE) # 모델 객체 생성, hidden_size1 = 512, hidden_size2 = 256, latent space = 2
optimizer = optim.Adam(VAE_model.parameters(), lr = 1e-3) # ADAM optimizer


def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction = 'sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE, KLD

# Loss function에 대한 코드는 논문의 Appendix C를 참고해야 한다.
# BCE 부분은 reconstruction error, 즉 loss function의 두 항 중에 MCMC estimate한 부분인데, 논문의 부록을 참고하면 디코더에 베르누이 MLP를 사용할 경우 BCE
# KLD부분 역시 부록을 참고하면 바로 equation을 알 수 있다.

```

▼ Full Code below

Google Colaboratory

 https://colab.research.google.com/drive/1tUZqOtD_-DML3uvfpdu0ZDgeauK3kasL?usp=sharing

